Plain JPA and Facades

This document provides information to set up simple JPA projects using NetBeans Wizards. For larger exercises, CA's and the semester project we suggest that you use our start code.

This document assumes a developer setup as explained in the instructions given for DAY-1.

JPA with NetBeans (my first java-JPA application)

This is a step by step introduction to how to get started with JPA. Use it as help for the first 1-2 JPA exercises, and after that, you are expected to know these fundamental steps.
Use the second part "JPA With a Facade" as a template for your future DataBase facades.

1) Create a new **plain** (Maven) java application

2) Create a new local MySQL database (on your Docker MySQL instance) (use Workbench or Netbeans)

3) Use the NetBeans `New File` Wizard and select **`Other`**… → In the window that pops up, select **`Persistence`** and now select → **`Entity Class`**

Type **`Book`** as Class Name, **`entity`** as Package, and change the primary key to **`Integer`**. Finally, make sure that "***Create Persistence Unit"*** is checked.

Press **next**.

In the next Wizard window select "**New Database Connection...**" in the dropdown.

In the Window that pops up, just press **Next** (assuming you have set up NetBeans as required)

In the next window:

Enter the name of the database you created in step two.

Enter **`dev`** as User Name and **`ax2`** as Password and select "**remember password**"

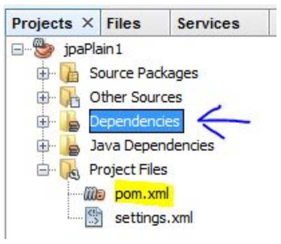
Press "**Test Connection**" and if everything is fine, press "**Finish**"

Finally, on the last wizard page, select **Drop and Create** for Table Generation Strategy, but make sure you understand the consequence of this, and *when you would have to change it*.

4) Include the MySQL Connector in your pom-file

Right-click Dependencies and type **`mysql-connector-java`** in the query field. Select the newest version of the driver.

Open the pom-file and verify that a reference to the driver has been added.

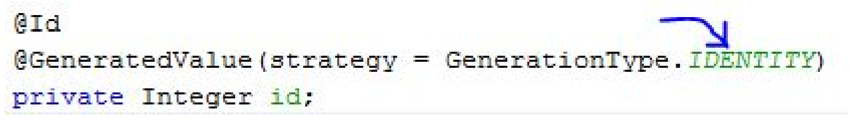


5) Change the generated Entity Class

Open the generated Entity Class **`Book.java`** and:

Delete the `hasCode()`, `equals(..)` and `toString()` methods. You can always generate them again, once your Class is completed.

Change the GenerationType used to annotate the id field into `GenerationType.IDENTITY`, as sketched below, to generate ids automatically using MySQL's AUTO_INCREMENT.

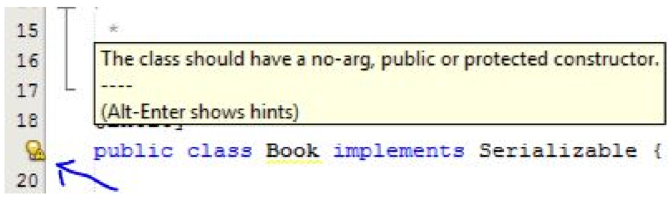


```
@Id
@GeneratedValue(strategy = GenerationType.IDENTITY)
private Integer id;
```

Make sure you understand why you are requested to use GenerationType.IDENTITY and not one of the alternatives.

Add a new private String field called **`author`** to `Book.java`, and provide a getter, setter and a constructor that takes the author as input (NOT the id)

IMPORTANT: One of the most common mistakes when starting with JPA is to forget that entity classes **must have a zero-argument constructor**. Because you just added a constructor with an argument, you no longer have the default zero-argument constructor.



The class should have a no-arg, public or protected constructor.

(Alt-Enter shows hints)

```
public class Book implements Serializable {
```

Locate the small icon next to the start of the class declaration. Right-click and you will see the problem as seen in this figure. Unfortunately, the icon is easy to miss, and JPA exceptions can be hard to read, so always remember to include a zero-argument constructor.

Hint if you get an error:

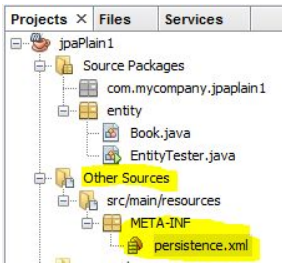
If you get an exception, similar to the one below, it's most likely because you are using a JDK newer than JDK8 which is the one we recommend, and the one used in the Tomcat docker image.

```
java.lang.NoClassDefFoundError: javax/annotation/Generated: javax.annotation.Generated
```

The recommended solution is to downgrade your JDK to 1.8, but alternatively you can add this dependency to your pom-file (but using a JDK other than 1.8 could cause other problems):

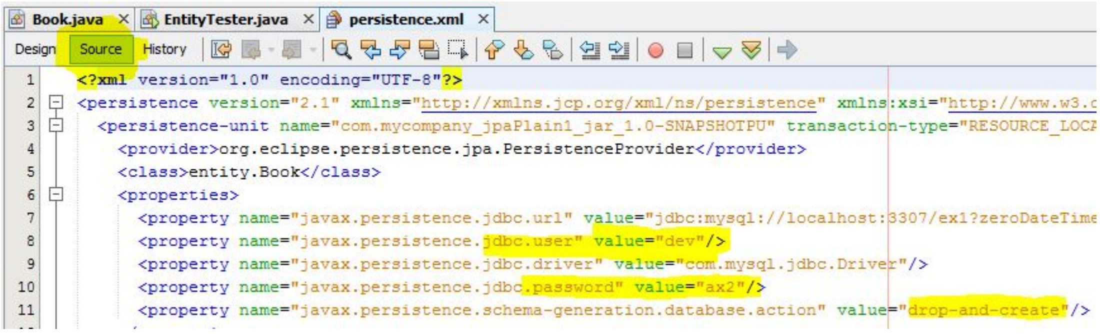
```
<dependency>
	<groupId>javax.annotation</groupId>
	<artifactId>javax.annotation-api</artifactId>
	<version>1.3.2</version>
</dependency>
```

6) Observe the generated persistence.xml file



Open the file persistence.xml and change the cumbersome generated name into **pu** which is easy to remember (in step 7).

Press the source tab (see figure below), to see the connection information stored for this connection. You will see later, how we can solve the problem with hard-coded credentials like this. For now, this is development so this is not a problem.



```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence" xmlns:xsi="http://www.w3.c
  <persistence-unit name="com.mycompany_jpaPlain1_jar_1.0-SNAPSHOTPU" transaction-type="RESOURCE_LOCA
    <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
    <class>entity.Book</class>
    <properties>
      <property name="javax.persistence.jdbc.url" value="jdbc:mysql://localhost:3307/ex1?zeroDateTime
      <property name="javax.persistence.jdbc.user" value="dev"/>
      <property name="javax.persistence.jdbc.driver" value="com.mysql.jdbc.Driver"/>
      <property name="javax.persistence.jdbc.password" value="ax2"/>
      <property name="javax.persistence.schema-generation.database.action" value="drop-and-create"/>
```

7) See the Entity Framework in "action"

Add a plain java-class **EntityTester.java** in the entity package (once you have learned JPA, we will create JUnit-tests to verify the behaviour).

Provide the class with a `public static void main(String[] args)` {} and add the following code to the method:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");
EntityManager em = emf.createEntityManager();
Book book1 = new Book("J.K. Rowling");
Book book2 = new Book("Georg R. R. Martin");
try {
        em.getTransaction().begin();
        em.persist(book1);
        em.persist(book2);
        em.getTransaction().commit();
        //Verify that books are managed and has been given a database id
        System.out.println(book1.getId());
        System.out.println(book2.getId());

         }finally{
                em.close();
         }
}
```

Execute this main method, and if everything was fine, go to Workbench and verify that:

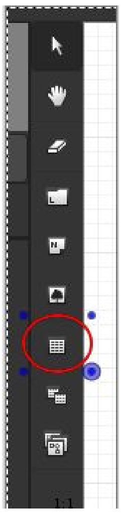
- The Book table was created
- Two rows were added to the Book-table

For an easier way

The above way to create entity classes are recommended in the beginning to ensure you understand the code that's being created.
It is possible though to create your entity classes based on an existing database.
This means that you can have a workflow that goes:

1. Model a database in workbench
2. Forward engineer the model to a real database with foreign keys (FK) and all
3. Make netbeans create all your entity classes with lots of template code (Huge time saver)
4. Clean up the generated classes if something is not to your liking.

How to do that?



1. In workbench: File -> New Model
2. Double click mydb icon to change the database/schema name
3. double click Add Diagram icon
4. In the left side menu choose “Place a new table icon”
5. Click inside the squared area to drop the table
6. Double click the table and change the table name and columns
7. For relationships to other tables (only possible if they have primary key defined) you can choose e.g. the icon: “1:n nonidentifying relationship” and click first one table then the other. Inspect to see what you get in terms of FKs
8. Now click Database -> Forward Engineer -> click next till you are finished
9. In your netbeans project create a new database connection to the new database
10. Right click project name and choose: new -> other -> Persistence -> Entity classes from database and choose the tables you need to create Entity classes from.

JPA With a Facade

There are many reasons to encapsulate your database interactions in a facade, the single most important one being that we can test all database interactions one single place.

This part continues with the previous exercise but provides a template from which you can design most of your database facades this semester.

1) Create a new plain Java class called `BookFacade` in a package `dbfacade`.
Observe how we pass in an EntityManagerFactory to the facade, so we can switch to a test database for our tests.

Paste in all the code below, and (important) add relevant java doc to all the methods (explaining the purpose, arguments and return values)

```
 private static EntityManagerFactory emf;
   private static BookFacade instance;

   private BookFacade() {}

   public static BookFacade getBookFacade(EntityManagerFactory _emf) {
       if (instance == null) {
           emf = _emf;
           instance = new BookFacade();
       }
       return instance;
   }

   public Book addBook(String author){
       Book book = new Book(author);
       EntityManager em = emf.createEntityManager();
       try{
           em.getTransaction().begin();
           em.persist(book);
           em.getTransaction().commit();
           return book;
       }finally {
           em.close();
       }
   }

   public Book findBook(int id){
        EntityManager em = emf.createEntityManager();
       try{
           Book book = em.find(Book.class,id);
           return book;
       }finally {
           em.close();
       }
   }
   public List<Book> getAllBooks(){
        EntityManager em = emf.createEntityManager();
       try{
           TypedQuery<Book> query =
                       em.createQuery("Select book from Book book",Book.class);
           return query.getResultList();
       }finally {
           em.close();
       }
   }
```

2) Testing the facade

For the more “serious” exercises you will do this semester you should always create two “local” databases. One for your ongoing development, and one for your JUnit-tests.
For this intro, however, we will use the database you created for the first part and just add a main method to the facade for a “manual” test.

Add a public static void main(..) to the BookFacade and paste in the following code:

```
EntityManagerFactory emf = Persistence.createEntityManagerFactory("pu");
BookFacade facade = BookFacade.getBookFacade(emf);
Book b1 = facade.addBook("Author 1");
Book b2 = facade.addBook("Author 2");
//Find book by ID
System.out.println("Book1: "+facade.findBook(b1.getId()).getAuthor());
System.out.println("Book2: "+facade.findBook(b2.getId()).getAuthor());
//Find all books
System.out.println("Number of books: "+facade.getAllBooks().size());
```

Right-click the file and select “Run File”. If everything was OK, use Workbench to verify that the two Books were added to the database.